Ex. No: 1 **DATE : 01.08.2023**

ABOUT THE PROJECT

A Campus Hall Booking System is a great idea to streamline event organization and optimize the usage of facilities on campus. Like the one which Indian Institute of Technology, Madras has: https://dost.iitm.ac.in/iitmdost/pages/halls-booking. It can make the process of reserving halls and rooms more efficient and accessible for students. Here are some features and benefits that could be included in such an application:

1.1 Features:

- **User Authentication:** Implement a secure login system using student email credentials to ensure only authorized users can make bookings.
- **Hall Availability Calendar:** Display a calendar that shows the availability of all halls and rooms, highlighting the already booked time slots.
- **Booking Form:** Provide a user-friendly booking form where students can select the desired hall, date, time slot, and purpose of the event.
- **Booking Confirmation and Reminders**: Send email confirmations and reminders to users about their booked events, reducing the chance of scheduling conflicts.
- **Booking Modification and Cancellation:** Allow users to modify or cancel their bookings within a specific time window to manage changing requirements.
- **Booking History:** Provide a history section where students can view their past and upcoming bookings.
- **Admin Panel:** Create an admin dashboard to manage hall availability, review booking requests, and handle any disputes or issues.

1.2 Benefits:

- **Time and Resource Optimization:** The system ensures efficient usage of campus halls and rooms by avoiding overlaps and preventing underutilization.
- **Increased Accessibility:** Allowing students to book halls online makes the process more accessible, especially for those who cannot physically visit the booking office.
- **Transparency:** The calendar view provides transparency, allowing students to see which halls are already booked, helping them make informed decisions.
- **Reduced Administrative Burden:** Automating the booking process reduces the administrative workload and minimizes the chances of manual errors.

- **Enhanced Event Planning:** Students can plan their events well in advance, giving them more time to prepare and promote their gatherings.
- **Improved Communication:** Automated email confirmations and reminders improve communication between the booking system and users.

By building this application, you can enhance the overall campus experience and contribute to a more organized and efficient event management system for the college community.

Ex. No: 2 **DATE : 14.08.2023**

FEASIBILITY REPORT

2.1 Introduction

This feasibility report assesses the viability of implementing a hall booking system. The report evaluates the legal, economic, technical, operational, and scheduling feasibility of the proposed project.

2.2 Operational Feasibility:

The Software enhances efficiency and accuracy as it records the data and proceeds with any action immediately. It can accommodate the organization's current and future needs as it grows and can be updated.

2.3 Technical Feasibility:

This software is feasible in technical sense. It can be integrated with other systems as the necessary tools are available. We only require a fair amount of time to acquire a good knowledge about how to use them.

2.4 Economic Feasibility:

Our software does not need any materialistic resources, so the cost factor is nil. The tools which we require to develop and implement this software are all open sources. The project is expected to be completed on or before the proposed deadline.

2.5 Scheduling Feasibility:

We planned our deadline to be at least 1 week before the affirmed deadline so that any change that incurs in short-notice will be rectified in the last few days or the rest of the time is enough to do fine tuning. The team will be capable of providing the primary features that will enable user to visualize the system.

2.6 Legal Feasibility:

We conforms that proposed project maintains the legal and ethical requirements without violating any user given rules. We assure that the project will protect the data from unauthorized users.

2.7 Conclusion:

According to the above feasible report, implementing the system should be beneficial for user by reducing their manual effort. The reduction of time consuming tasks results in increased efficiency. As a result, the project is entirely feasible.

Ex. No: 3 **DATE : 21.08.2023**

REQUIREMENTS ENGINEERING

3.1 User Requirements:

● Login Page

A login page each for Student users and the admin. Captcha Included

● New User Registration

Registration Portal for students. Authorized using Email and E-Identity Card. Username and Password chosen by Student.

● Admin Dashboard

Admin Dashboard for each admin.

● Calendar

Dedicated Calendar that shows status of each hall whether approved, free, pending or booked. After checking with this user can go for new request.

● Booking Form

Booking form as per University standards asking for details and reason for booking.

● Halls

List Halls in the Home page for which can be booked.

● Halls Details

When the user clicks the Hall name, Details of Hall including pictures and address of the Hall is shown.

● Guide for Booking

User Guide for booking halls.

● Booking Form

Booking form for users which is needed for hall booking.

● New Request

New request option after login where users can choose the hall to book and date.

● User Dashboard

Shows the status of the requests and approved requests.

● Approved Request PDF

PDF which can be downloaded after the approval

● Status Tracking

Status of Hall indicated with colour.

● Requests Page

Shows the request to the admin. Admin can verify the details and reason for booking.

● Email System

Email system where email is sent to the user when request is approved and to the admin when a new request is initiated.

● New Hall Addition

New hall addition system which can be accessed by the dean office. Details of the hall are added (Hall name, Hall Admin In-charge, Hall picture, Hall Address).

3.2 System Requirement Specification:

3.2.1 Functional requirements :

1. Login page :

Login page should get the following information from the user :

- Email ID
- Password
- Captcha

It should also have instruction for new users who have not registered yet and also a link to Forgot Password.

2. Student Registration Page :

The Registration page for students should get the following information:

- Applicant Name
- Email ID

- Department (which the student belongs to)
- Password
- Confirm Password
- Captcha

It should also have instructions for users who have already registered.

3. Department In-charge Registration Page :

The Registration page for department in-charge should get the following information :

- Applicant Name
- Email ID
- Department (in-charge of)
- Password
- Confirm Password
- Captcha

The user should actually get a confirmation from the Dean in order to be able to gain access of the chosen department.

4. Forgot Password :

An option to change the password in case the user forgot the password. The user should be verified by sending an OTP to their email address. If valid then user can change his/her password. The following information are required :

- New Password
- Confirm password

5. Student User Dashboard :

The Students User Dashboard should display the applicant's name along with a profile picture, there should be an option to change the profile picture too. The Dashboard should have the following tabs :

- Home
- Hall Details
- Booking Status
- Hall Booking
- Help
- Logout Option

The Home tab should be the default tab that should be loaded after logging in.

6. Recommendation of Halls :

The home tab should contain the guidelines to book an hall. It should also show a list of top 5 of the total halls. These halls should be recommended based on the previous bookings of the user.

If the user is new and haven't booked any halls yet, then the halls are recommended based on their home department.

7. Booking Status Page :

This page should show all the bookings made by the user in the form of cards. Each card should have the following information :

- Hall Name
- Date
- Time (from – to)
- Department (in which the hall is present)
- Affiliated Department or Club
- Reason
- Submitted time.
- Status (Approved, Pending, Rejected)

The bookings should be colorized based on their status, Red – Rejected, Green – Approved, Orange – Pending.

8. Approved Request PDF :

In case of approved bookings, there should be option to print the approval as a document in pdf format. The document should contain the following information :

- Hall Name
- Department
- Booked by (Student name)
- Date
- Time (from – to)
- Affiliated department or club
- Reason
- Statement from the respective Department In-charge.

The document should be like a certificate that shows the booking was valid and was verified by the respective in-charge personnel.

9. Hall Booking Form :

Booking form as per University standards asking for details and reason for booking. The booking form collects the following information :

- Department (The hall to booked)
- Hall to be booked

- Affiliated Department or club
- Date
- Time From
- Time To
- Reason for Booking
- Capacity of audience expected

Since every hall is linked to some department, if the department option is chosen, then only the halls in that particular department should be shown in the halls option.

10. Logout Option :

A logout option for all users with confirmation before logging out.

11. Availability Calendar :

A dedicated calendar that shows each booking made with respective colour code based on their status (Green - Approved, Red - Rejected, Orange - Pending). There should be an option to view the calendar in monthly, weekly and daily format. Also there should be an option to filter the bookings based on the halls.

12. Department In-charge Dashboard :

The dashboard of department in-charges should have the following tabs :

- Home
- Pending Request
- Logout

13. Home page of Department In-charge :

This page should contain the list of all halls which are under their control, the instructions to handle booking request and also an option to ‘Add new hall’.

14. New Hall Addition :

There should be an option to add new hall in the department, the following information should be collected when creating a new hall :

- Hall Name
- Total capacity
- Images
- Description (about the hall)

These information are required to display in the Hall Details Page.

15. Pending Request Page :

The Pending Request Page of the Dashboard shows the all the requests (Approved, Pending and Rejected). There should be a filter to choose between these requests. These request should be colorized based on their status (Red – Rejected, Green - Approved, Orange – Pending).

16. Main Admin Dashboard :

The dashboard of the Main Admin (Dean) has options to approve requests which require Dean approval and also an option to 'Add new department'.

17. New Department Addition :

The Main Admin also has an option to add a new department.

18. Home Page :

The home page should have a nav bar with links to the following pages :

- Calendar
- Admin
- Hall Details
- About
- Link to Login | Register

It should have instructions on how to use the website and a list of all halls.

19. Hall Details Page :

When an hall is clicked all its details are shown including total capacity and in-charge personnel. The Hall Details Page should contain the following information :

- Hall Name
- Department
- In-charge personnel
- Total capacity
- Images
- Description about the hall.

20. Guide for booking :

The set of instructions to book a hall right from registering to getting the approval PDF.

21. Email System :

Email System is required to send emails in the following cases :

- Forgot Password

- Notifications

3.2.2 Non Functional Requirements :

1. Scalability :

The website should work properly even when multiple users are logged in.

2. Responsive :

The website should be responsive and should designed such that it can be used through any devices (PC, Mobile, Tablets, etc).

3. Integration :

The system should have the ability to extend its requirements.

4. Maintainability :

The system should be easily maintainable to allow for additional upgrades that can be implemented in the future.

5. Capacity :

The system should have minimum storage requirements. Past data of booked halls can be deleted after a certain period of time to avoid memory build up.

3.3 User to System Requirement:

1. User Requirement: User Dashboard - Shows the status of the requests and approved requests.

System Requirement: The Students User Dashboard should display the applicant's name along with a profile picture, there should be an option to change the profile picture too. The Dashboard should have the tabs – Home, Hall Details, Booking Status, Hall Booking, Help and Logout Option. The Home tab should be the default tab that should be loaded after logging in.

2. User Requirement: New hall addition system which can be accessed by the dean office. Details of the hall are added (Hall name, Hall Admin In-charge, Hall picture, Hall Address).

System Requirement: There should be an option to add new hall in the department, the following information should be collected when creating a new hall - Hall Name, Total capacity, Images, Description (about the hall). These information are required to display in the Hall Details Page.

3.4 Requirement Discovery

The Above Requirements are collected by performing several activities interacting with stakeholders like Students, Hall incharge etc., in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.

At first our team created a Google Form https://forms.gle/9QNT1sCP3Kabz5Sc6 to collect a suggestions and requirements from the Student through which we can able see the whether the project is feasible or not. And also we gathered the User Requirement through which we can develop our system that could satisfy the user expectation.

Next Step we conducted some interview to the hall in chargers and sanitary inspectors who are located near the red building to gather their suggestion on the project and their requirements. Using this we came to a clarity about how the existing system for booking a hall works.

At Last, we performed a discussion with Staffs at Ramanujam Computing Center and Officers at E-Governance Office to see whether the project is technically feasible and asked about tech stack that would better for the project.

Ex. No: 4 **DATE : 28.08.2023**

REQUIREMENT ANALYSIS

4.1 Requirement Validation

I. During our initial requirements analysis, we doesn't have a plan for add new department page, but later realized that there need to have a separate add department page. This would be shown to dean who has authority to add new department.

II. At first we misunderstood the hall booking procedure. We designed the system to book a slot without checking whether the slot is free or not. We assumed that admin can approve the checking by seeing the vacancy. But later we changed this, where the system check the vacancy of the slot before updating the database.

III. At first we didn't think of the feature of cancel a booking, but upon discussing with our stakeholders (our colleagues) we understood that this feature is important and hence decided to include it to our requirements.

Ex. No: 5 **DATE : 05.09.2023**

SYSTEM REQUIREMENTS DOCUMENT

5.1 Preface:

This Software Requirements Specification (SRS) document is intended for users of the Campus Hall Booking System and people evaluating its potential use. This is the introductory version Campus Hall Booking System.

5.2 Introduction:

The main objective of the system is to reduce manual maintenance of hall booking procedure and facilitate this through by developing a software. The System provides an interface to the hall in charge to accept or deny the booking towards the hall by viewing the past booking on the hall to ensure whether the slot free or not. The System also provides an interface to the student to book an hall or to track past booking.

The launch of the application will close the gap in terms of quality of life and sow an awareness about conservation of resources among all the young minds. This application is aimed to help the campus and to create a sense of solidarity through modern techniques.

5.3 Glossary:

User – It Specifies the Student of the Campus

Admin – It specifies the person who is the incharge of the hall

HBMS – Hall Booking Management System

5.4 User Requirement Definition:

● Login Page

A login page each for Student users and the admin. Captcha Included

● New User Registration

Registration Portal for students. Authorized using Email and E-Identity Card. Username and Password chosen by Student.

● Admin Dashboard

Admin Dashboard for each admin.

● Calendar

Dedicated Calendar that shows status of each hall whether approved, free, pending or booked. After checking with this user can go for new request.

● Booking Form

Booking form as per university standards asking for details and reason for booking.

● Halls

List Halls in the Home page for which can be booked.

● Halls Details

When the user clicks the Hall name, Details of Hall including pictures and address of the Hall is shown.

● Guide for Booking

User Guide for booking halls.

● Booking Form

Booking form for users which is needed for hall booking.

● New Request

New request option after login where users can choose the hall to book and date.

● User Dashboard

Shows the status of the requests and approved requests.

● Approved Request PDF

PDF which can be downloaded after the approval

● Status Tracking

Status of Hall indicated with color.

● Requests Page

Shows the request to the admin. Admin can verify the details and reason for booking.

● Email System

Email system where email is sent to the user when request is approved and to the admin when a new request is initiated.

● New Hall Addition

New hall addition system which can be accessed by the dean office. Details of the hall are added (Hall name, Hall Admin In-charge, Hall picture, Hall Address).

5.5 System Architecture:

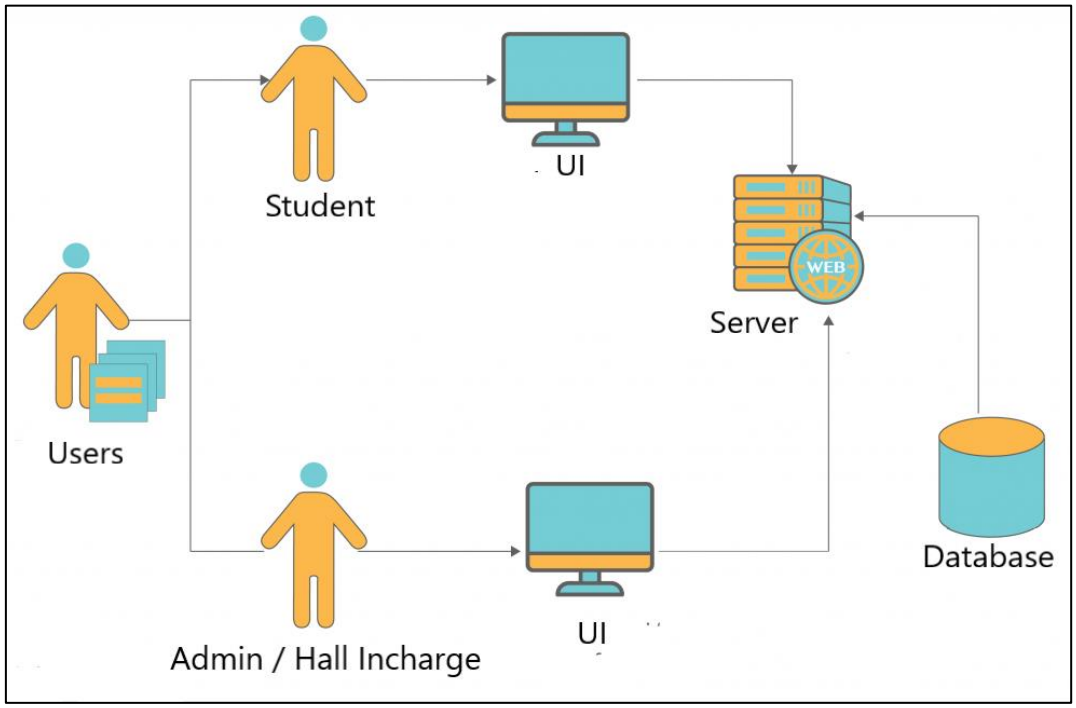

Figure 5.1 System Architecture

5.6 System Requirement Specification:

Functional requirements :

1) Login page :

Login page should get the following information from the user :

- Email ID
- Password
- Captcha

It should also have instruction for new users who have not registered yet and also a link to Forgot Password.

2) Student Registration Page :

The Registration page for students should get the following information:

- Applicant Name
- Email ID
- Department (which the student belongs to)
- Password
- Confirm Password
- Captcha

It should also have instructions for users who have already registered.

3) Department In-charge Registration Page :

The Registration page for department in-charge should get the following information :

- Applicant Name
- Email ID
- Department (in-charge of)
- Password
- Confirm Password
- Captcha

The user should actually get a confirmation from the Dean in order to be able to gain access of the chosen department.

4) Forgot Password :

An option to change the password in case the user forgot the password. The user should be verified by sending an OTP to their email address. If valid then user can change his/her password. The following information are required :

- New Password
- Confirm password

5) Student User Dashboard :

The Students User Dashboard should display the applicant's name along with a profile picture, there should be an option to change the profile picture too. The Dashboard should have the following tabs :

- Home
- Hall Details
- Booking Status
- Hall Booking
- Help
- Logout Option

The Home tab should be the default tab that should be loaded after logging in.

6) Recommendation of Halls :

The home tab should contain the guidelines to book an hall. It should also show a list of top 5 of the total halls. These halls should be recommended based on the previous bookings of the user.

If the user is new and haven't booked any halls yet, then the halls are recommended based on their home department.

7) Booking Status Page :

This page should show all the bookings made by the user in the form of cards. Each card should have the following information :

- Hall Name
- Date
- Time (from – to)
- Department (in which the hall is present)
- Affiliated Department or Club
- Reason
- Submitted time.
- Status (Approved, Pending, Rejected)

The bookings should be colorized based on their status, Red – Rejected, Green – Approved, Orange – Pending.

8) Approved Request PDF :

In case of approved bookings, there should be option to print the approval as a document in pdf format. The document should contain the following information :

- Hall Name
- Department
- Booked by (Student name)
- Date
- Time (from – to)
- Affiliated department or club
- Reason
- Statement from the respective Department In-charge.

The document should be like a certificate that shows the booking was valid and was verified by the respective in-charge personnel.

9) Hall Booking Form :

Booking form as per University standards asking for details and reason for booking. The booking form collects the following information :

- Department (The hall to booked)
- Hall to be booked
- Affiliated Department or club
- Date
- Time From
- Time To
- Reason for Booking
- Capacity of audience expected

Since every hall is linked to some department, if the department option is chosen, then only the halls in that particular department should be shown in the halls option.

10) Logout Option :

A logout option for all users with confirmation before logging out.

11) Availability Calendar :

A dedicated calendar that shows each booking made with respective colour code based on their status (Green - Approved, Red - Rejected, Orange - Pending). There should be an option to view the calendar in monthly, weekly and daily format. Also there should be an option to filter the bookings based on the halls.

12) Department In-charge Dashboard :

The dashboard of department in-charges should have the following tabs :

- Home
- Pending Request
- Logout

13) Home page of Department In-charge :

This page should contain the list of all halls which are under their control, the instructions to handle booking request and also an option to ‘Add new hall’.

14) New Hall Addition :

There should be an option to add new hall in the department, the following information should be collected when creating a new hall :

- Hall Name
- Total capacity
- Images
- Description (about the hall)

These information are required to display in the Hall Details Page.

16) Pending Request Page :

The Pending Request Page of the Dashboard shows the all the requests (Approved, Pending and Rejected). There should be a filter to choose between these requests. These request should be colorized based on their status (Red – Rejected, Green - Approved, Orange – Pending).

17) Main Admin Dashboard :

The dashboard of the Main Admin (Dean) has options to approve requests which require Dean approval and also an option to ‘Add new department’.

18) New Department Addition :

The Main Admin also has an option to add a new department.

19) Home Page :

The home page should have a nav bar with links to the following pages :

- Calendar
- Admin
- Hall Details
- About
- Link to Login | Register

It should have instructions on how to use the website and a list of all halls.

20) Hall Details Page :

When an hall is clicked all its details are shown including total capacity and in-charge personnel. The Hall Details Page should contain the following information :

- Hall Name
- Department
- In-charge personnel
- Total capacity
- Images
- Description about the hall.

21) Guide for booking :

The set of instructions to book a hall right from registering to getting the approval PDF.

22) Email System :

Email System is required to send emails in the following cases :

- Forgot Password
- Notifications

Non Functional Requirements :

1) Scalability :

The website should work properly even when multiple users are logged in.

2) Responsive :

The website should be responsive and should designed such that it can be used through any devices (PC, Mobile, Tablets, etc).

3) Integration :

The system should have the ability to extend its requirements.

4) Maintainability :

The system should be easily maintainable to allow for additional upgrades that can be implemented in the future.

5) Capacity :

The system should have minimum storage requirements. Past data of booked halls can be deleted after a certain period of time to avoid memory build up.

5.7 System models:

Use Case Diagram

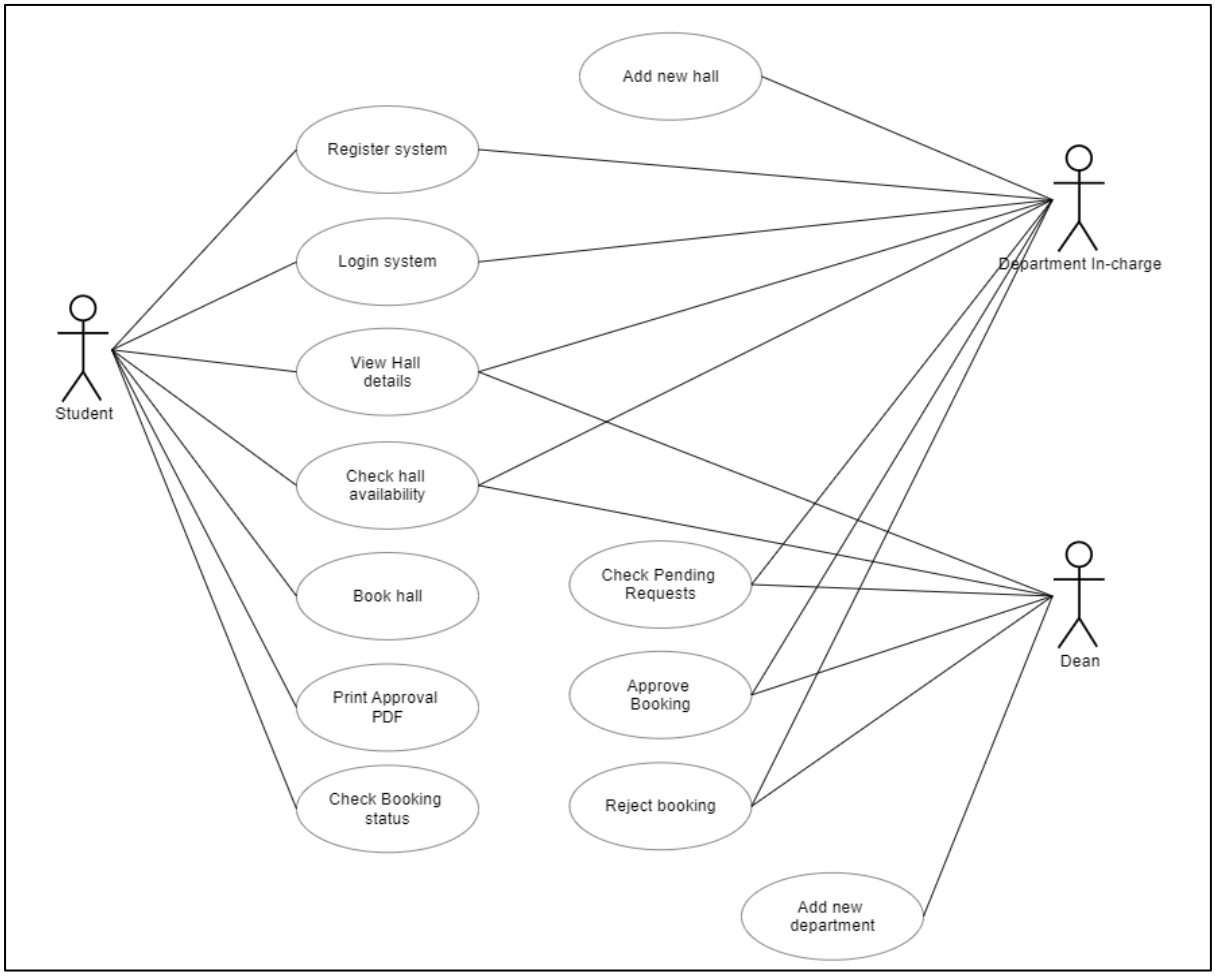

Figure 5.2 : Use Case Diagram

Sequence Diagram

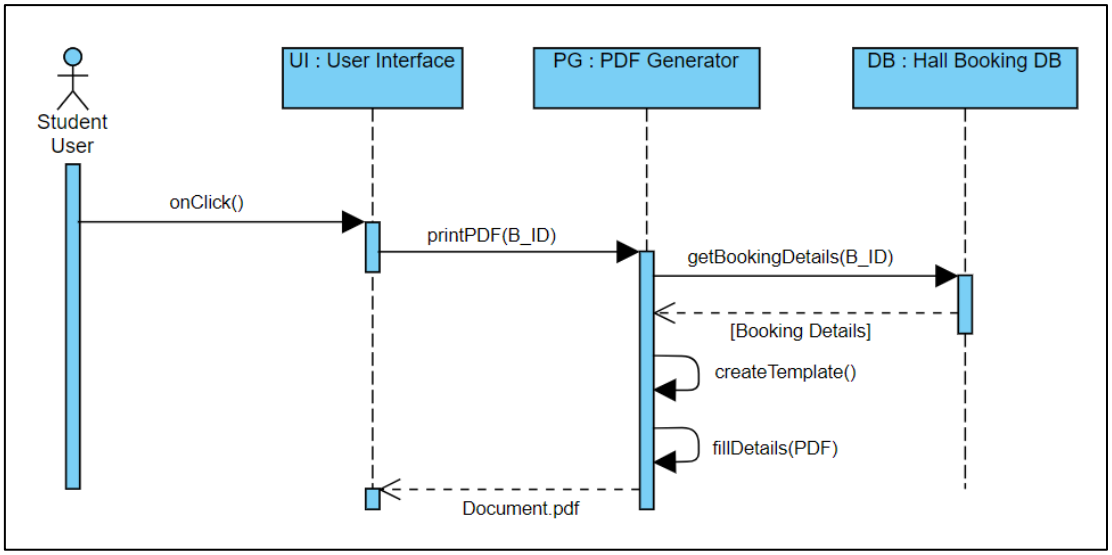

Figure 5.3 :Sequence Diagram for Print Approval PDF

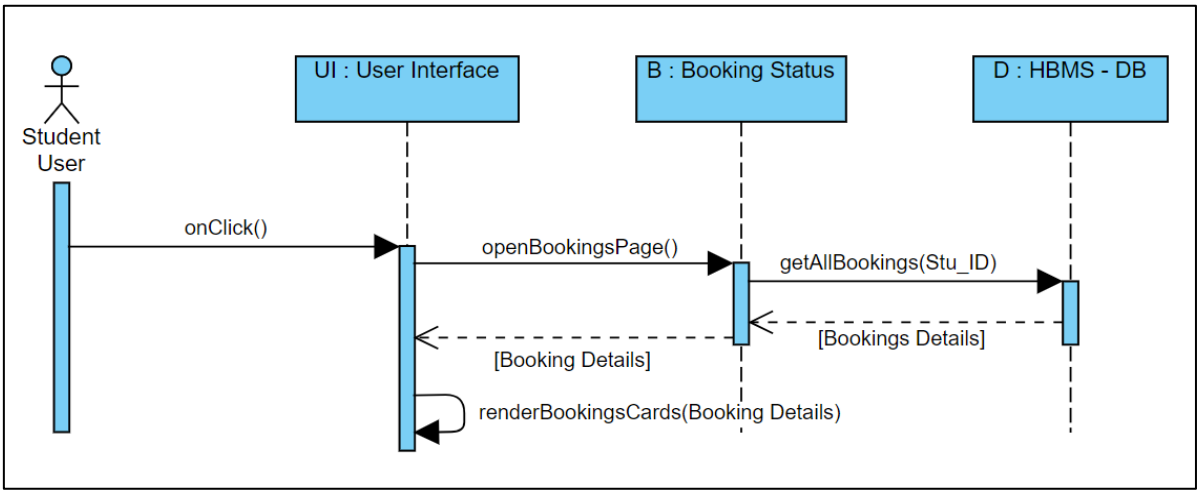

Figure 5.4 :Sequence Diagram for Check Booking Status

Class Diagram

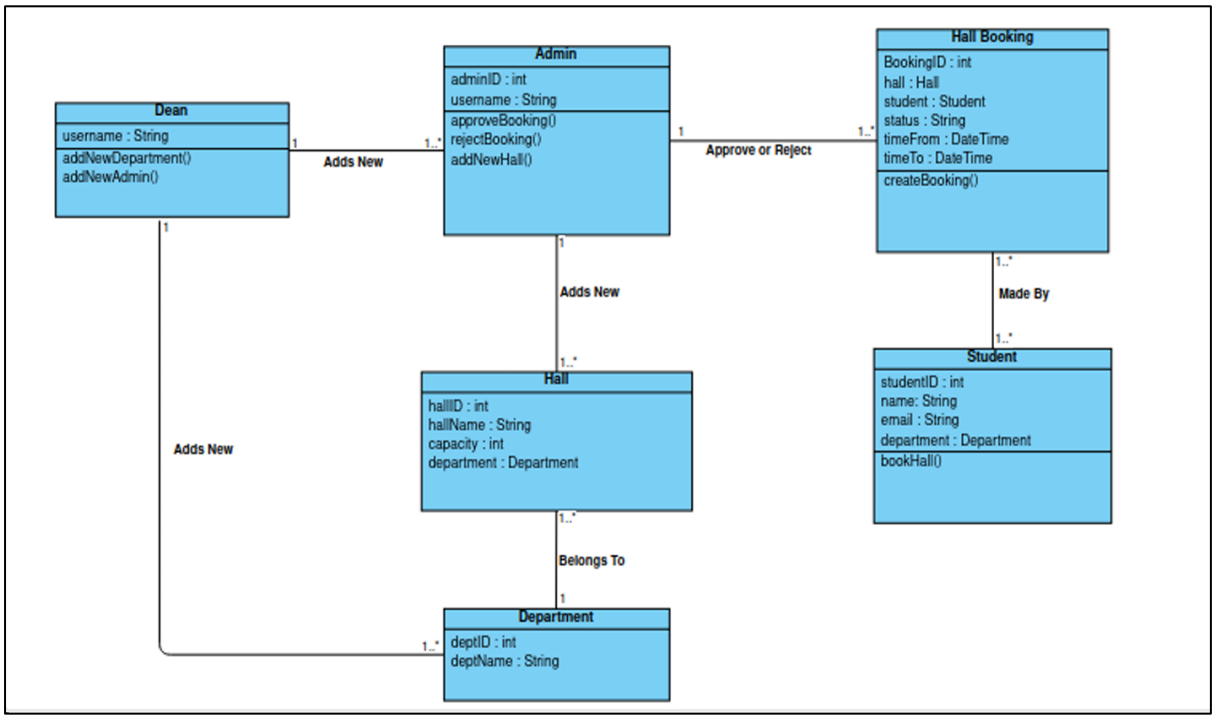

Figure 5.5 : Class Diagram

Activity Diagram

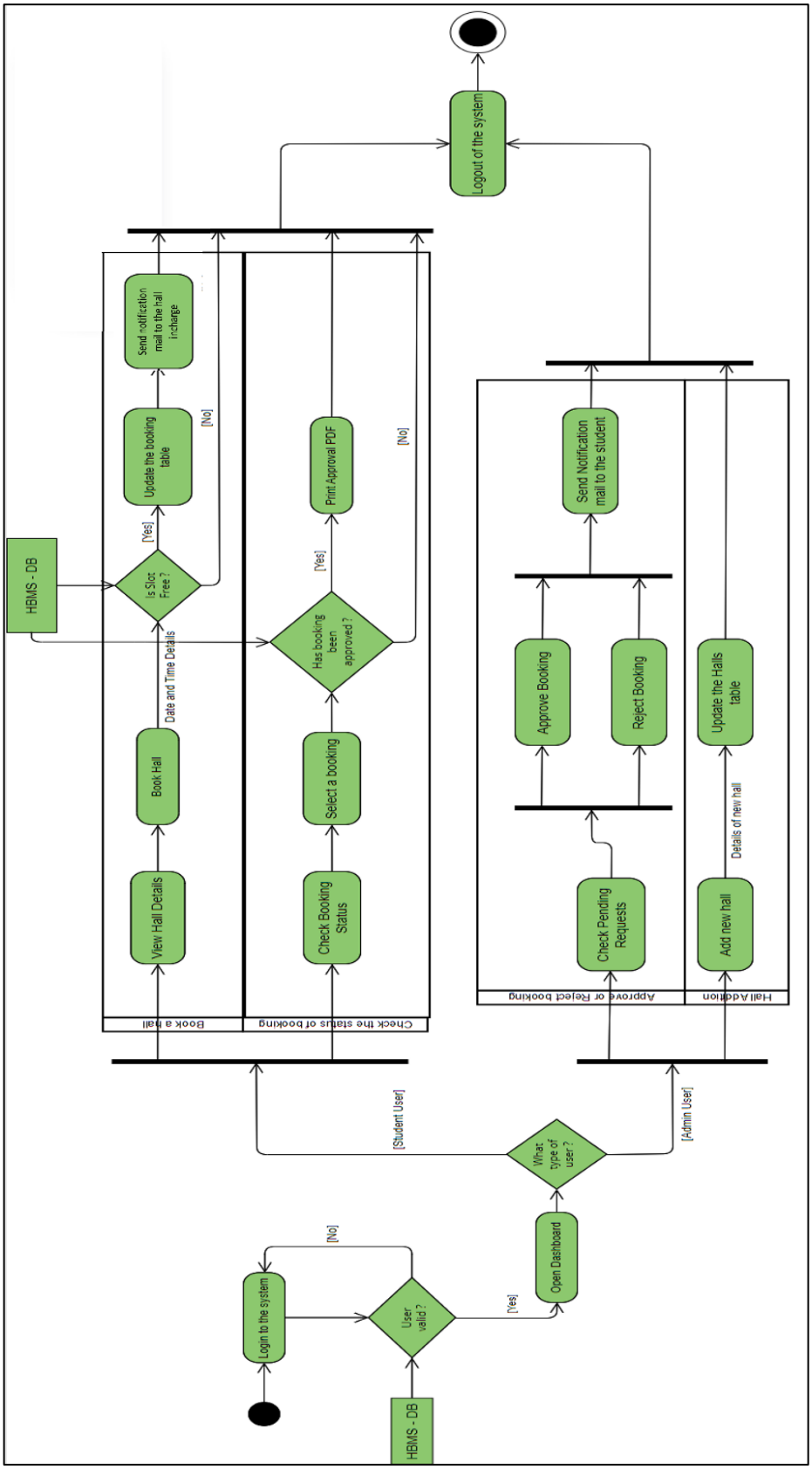

Figure 5.6 : Activity Diagram

State Diagram

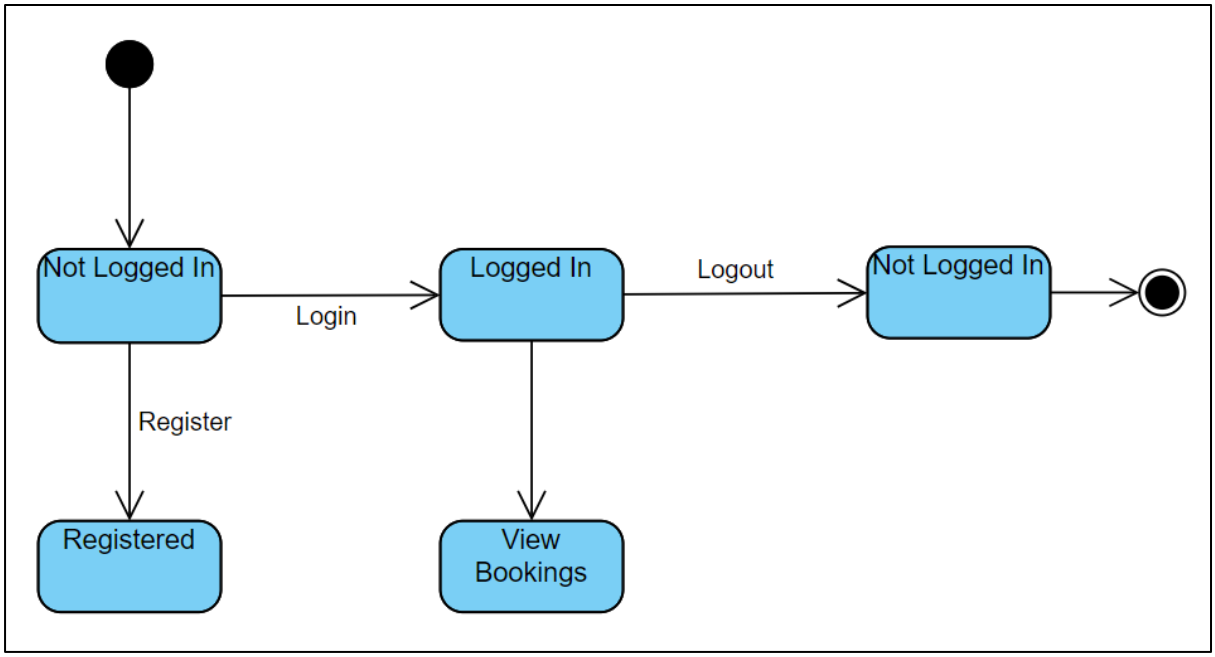

Figure 5.7 : State Diagram for User Entity

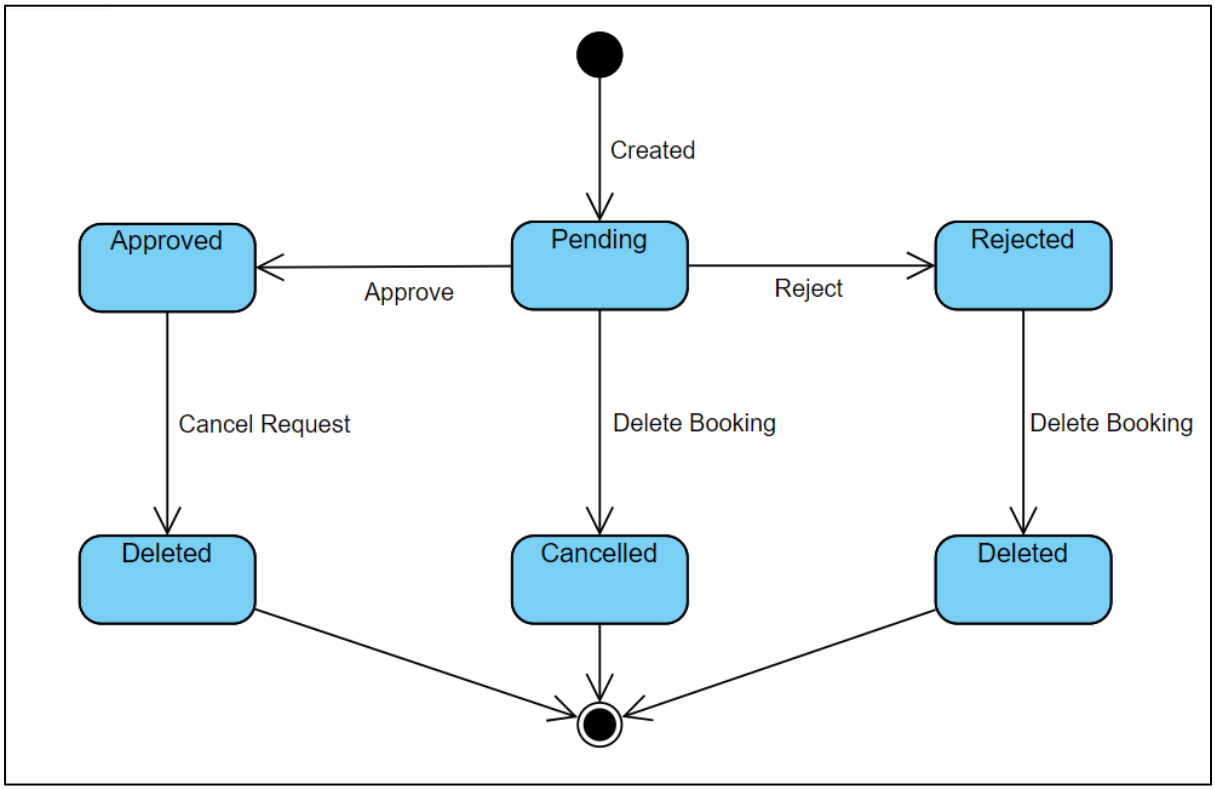

Figure 5.8 : State Diagram for Booking Entity

5.8 System Evolution:

The idea of Campus Hall Booking System has varied possibilities with wide scope, to the future it can be integrated with additional features. Also, it can be further developed to promote with exploring more halls and new hall can be added. In future, with desired support the system can be utilized for all other three campus. Our application involves a series of improvements and updates over time to meet changing user needs, technological advancements, and business requirements.

5.9 Appendices:

1) Minimum hardware requirements

1) RAM - at least 4GB
2) CPU - A multi-core processor for handling concurrent tasks efficiently.

2) Software Tools Used

1. MongoDB is a NoSQL document database. It stores data in a type of JSON format called BSON

2. Express.js is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features.

3. React is a free and open-source front-end JavaScript library for building user interfaces based on components.

4. Node.js is a backend JavaScript runtime environment, runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser.

3) Network requirements

High-speed internet connectivity is essential to support data transfers. A stable internet connection is required to use the application in order to communicate with the hosted database.

5.10 Indexes:

LIST OF FIGURES

Ex. No: 6 **DATE : 18.09.2023**

SOFTWARE ARCHITECTURE

The System follows the client server model where client request for the service to the server and server provides the service using the database for information. The Web Servers offers the services to other sub-systems. The clients call on the services offered by servers and utilize them.

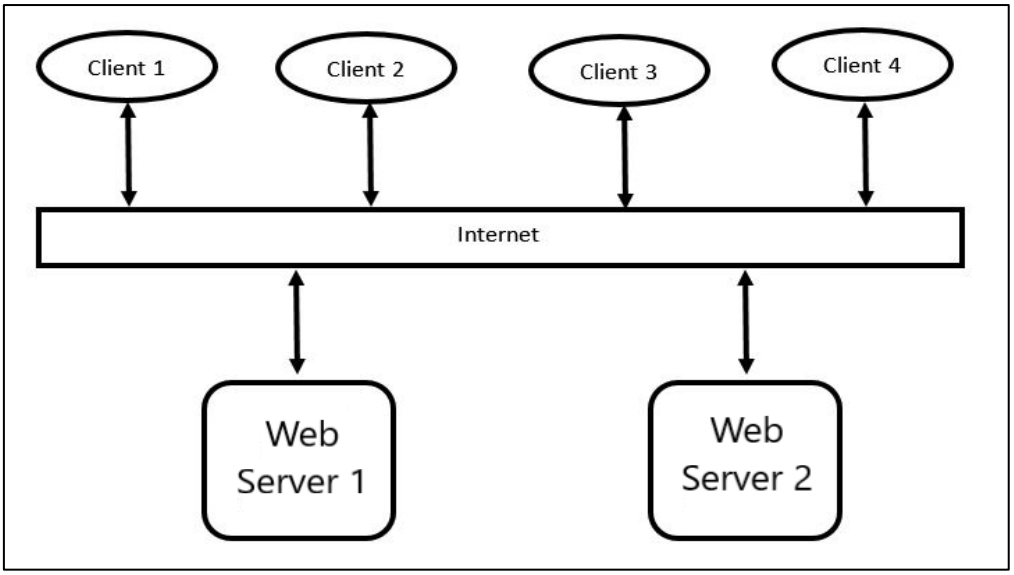

Figure 6.1: A pipeline model of a hall booking system

The sub-systems implemented based on the functional oriented pipelining where data flows from one to another and is transformed as it moves through the sequence. outputs. Data flows from one to another and is transformed as it moves through the sequence. Each processing step is implemented as a transform. The system is a Data-processing applications.

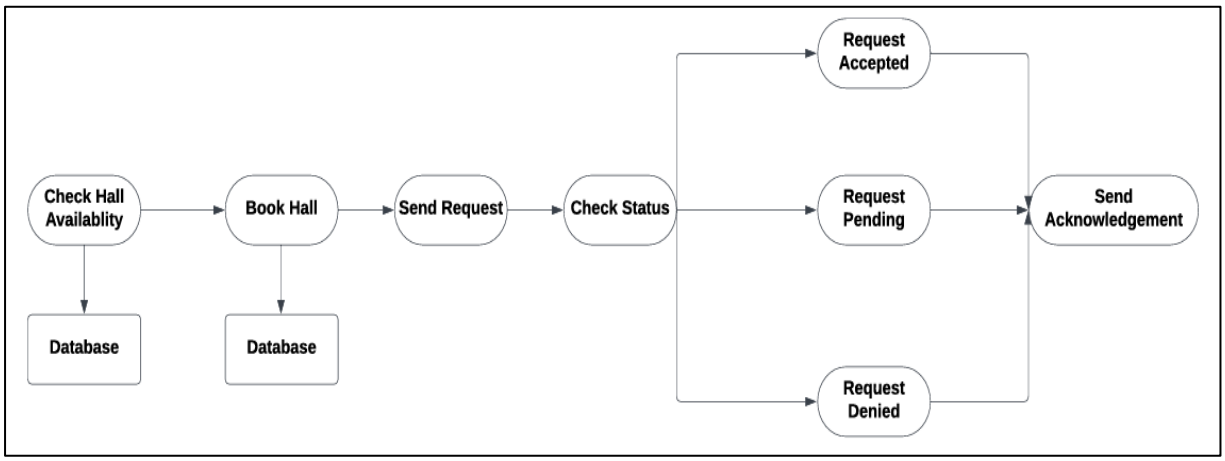

Figure 6.2: A pipeline model of a hall booking system

Ex. No: 7 **DATE : 25.09.2023**

SOFTWARE DESIGN

7.1 Use Case Diagram

The use case diagram of this project consists of three actors – Student, Department In-charge and Dean. The activities performed by student actors are login, view hall details and book a hall. The activities performed by Department In-charge actors are login, add a new hall, response to the booking and logout. Other than this the dean can add a new department and assign a staff to the hall.

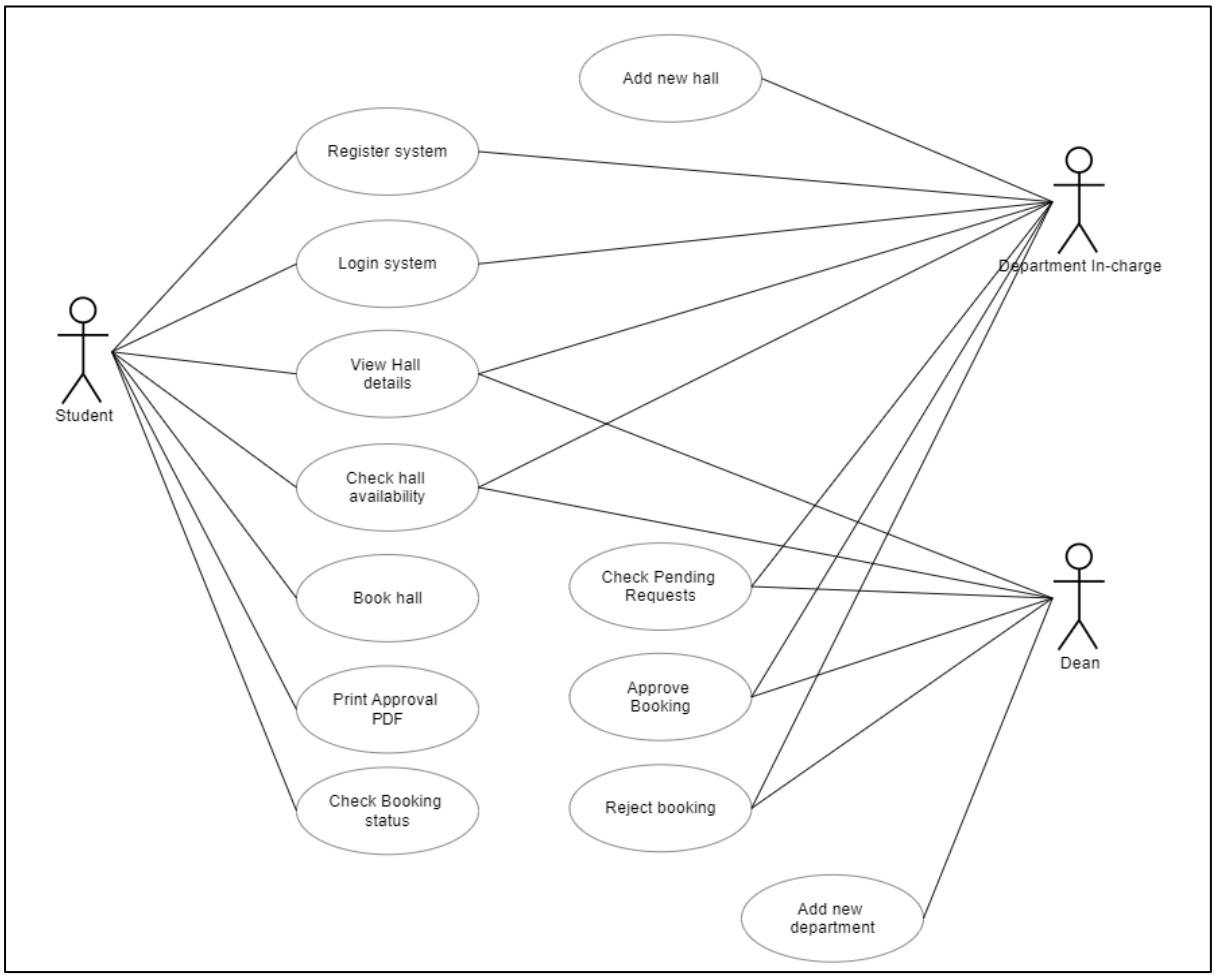

Figure 7.1 : Use Case Diagram

7.2 Sequence Diagram

1. Check Booking Status :

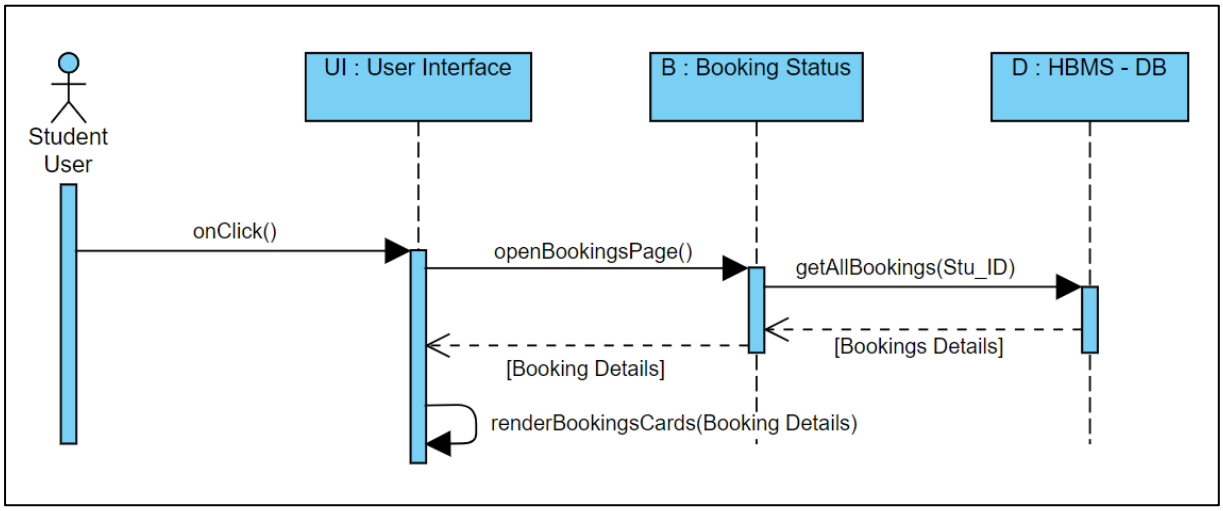

Figure 7.2 : Sequence Diagram for Check Booking Status

The Student User clicks a button saying "Booking Status" on the dashboard. This invokes a function to open the Booking Page. Which then automatically invokes another function to get all the bookings from the database through an API. The collected booking details are then rendered as separate cards on the frontend page.

2. Admin Approve Booking :

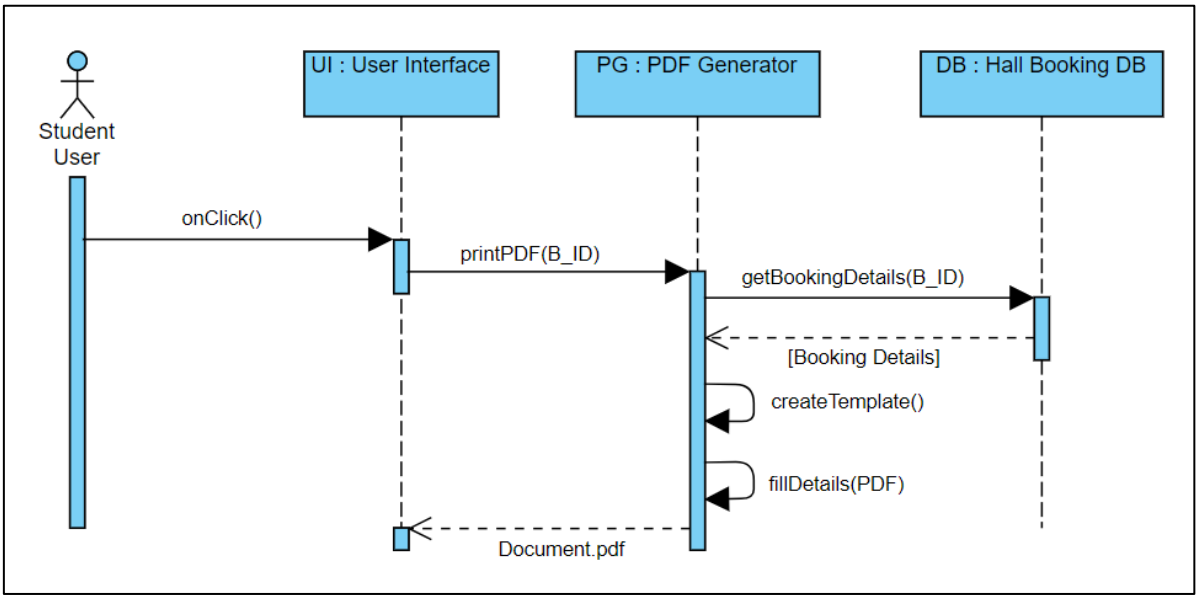

Figure 7.3 : Sequence Diagram for Print Approval PDF

The Admin opens the Pending Requests Page by clicking a button in their dashboard, which opens the page while collecting all the bookings corresponding to the particular Admin_ID and renders each Booking in a card component which contains an Approve Button. When Admin clicks the approve button, the respective Booking_ID is updated and status is changed to “Approved”. If the database was successfully updated, display the message to the user. Then create a mail template with the details of tha booking and send that mail to the email address of the Student user who made that booking.

7.3 Class Diagram

A class diagram is a visual representation of the structure and relationships of the classes within a system. In the context of a Campus Hall Booking System, the class diagram has Six different classes:

1. Dean Class:
    - Attributes: Username
    - Operations: addNewDepartment(), addNewAdmin()
2. Admin Class:
    - Attributes: AdminID, Username
    - Operations: approveBooking(), rejectBooking(), addNewHall()
3. Hall Booking Class:
    - Attributes: bookingID, hall, studentID, status, timeFrom, timeTo
    - Operations: creatingBooking()
4. Hall Class:
    - Attributes: hallID, hallName, capacity, department
5. Student Class:
    - Attributes: student ID, name, department
    - Operations:bookHall()
6. Department Class:
    - Attributes: departmentID, departmentName

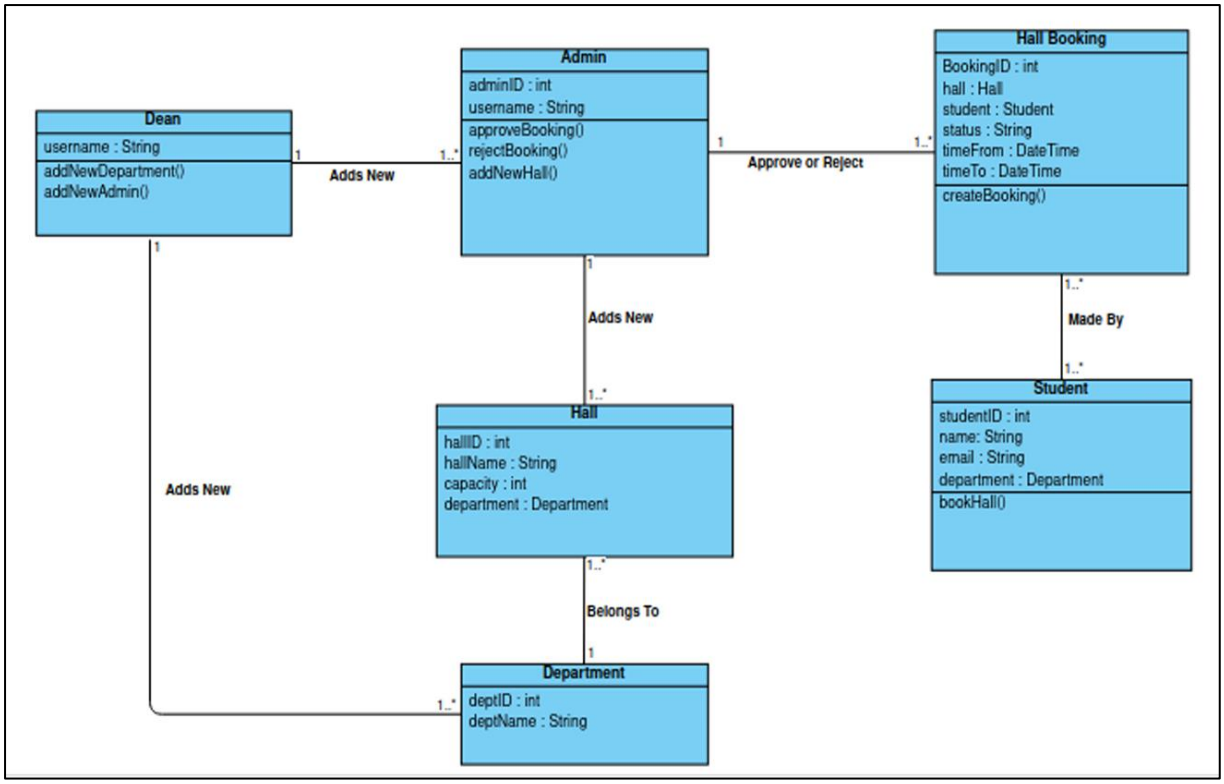

Figure 7.4 : Class Diagram

7.4 Activity Diagram

An activity diagram visually represents the flow of activities within a system. In the context of a Campus Hall Booking System with students and admins as users, the activity diagram starts with an initial node representing the beginning of the process.

After that the user where validated and corresponding menu where shown. The student initiates the process by requesting a hall booking. The system prompts the student to provide details such as date, time, purpose, and preferred hall. It Books a slot until the slot is free.

The system forwards the hall booking request to the admin for review. The admin evaluates the request and decides whether to approve or reject it. If the admin approves the booking request, then the system generates a confirmation for the student and notify Student. If the admin rejects the booking request, then the system notifies the student and provides relevant details. The process concludes with an end node.

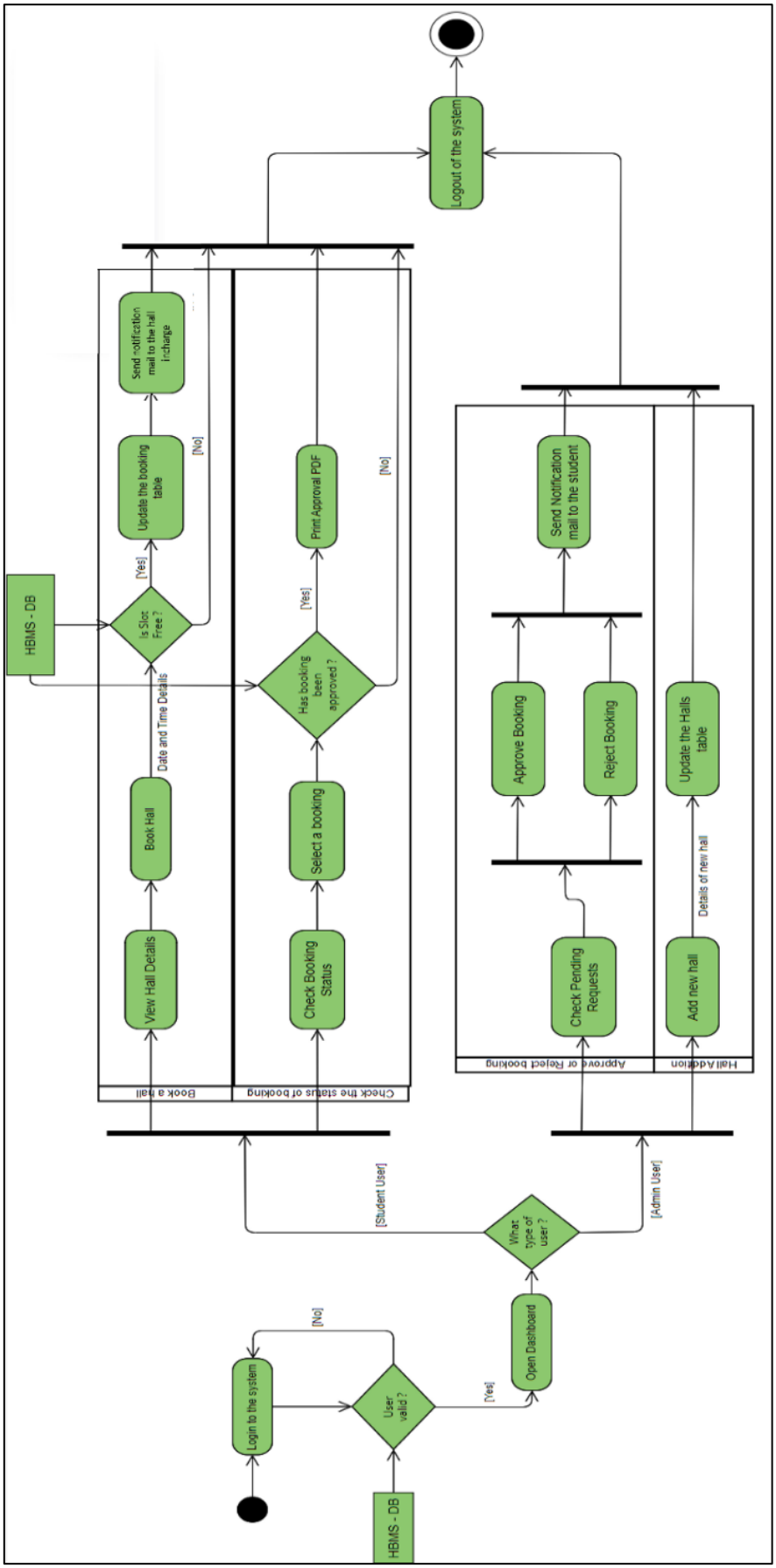

Figure 7.5 : Activity Diagram

7.5 State Diagram

Student User : This the state diagram for the Student User Entity. Initially, when the student is new to our website, he/she is Not Logged In state. If the student is not registered yet, then he/she can register and go the Registered state. If the student is already registered, then he/she can login using their credentials and go to Logged In state. Once logged in, the student can View his/her previous bookings and then finally logout.

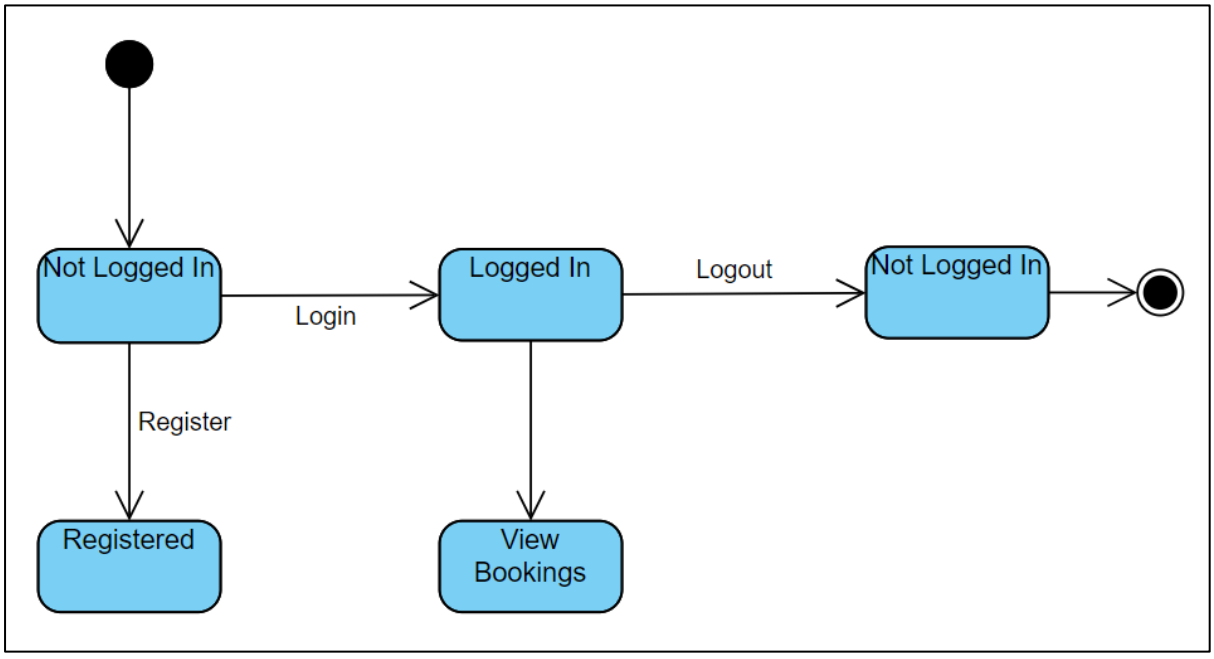

Figure 7.5 : State Diagram for User Entity

Booking : Once a booking has been created, it will always be assigned the status of "Pending" by default. Then the respective hall incharge can either approve or reject the booking. In both cases, the status attribute of the booking entity is changed, but the booking still exists. It is only deleted after the student user who made the booking deletes it.

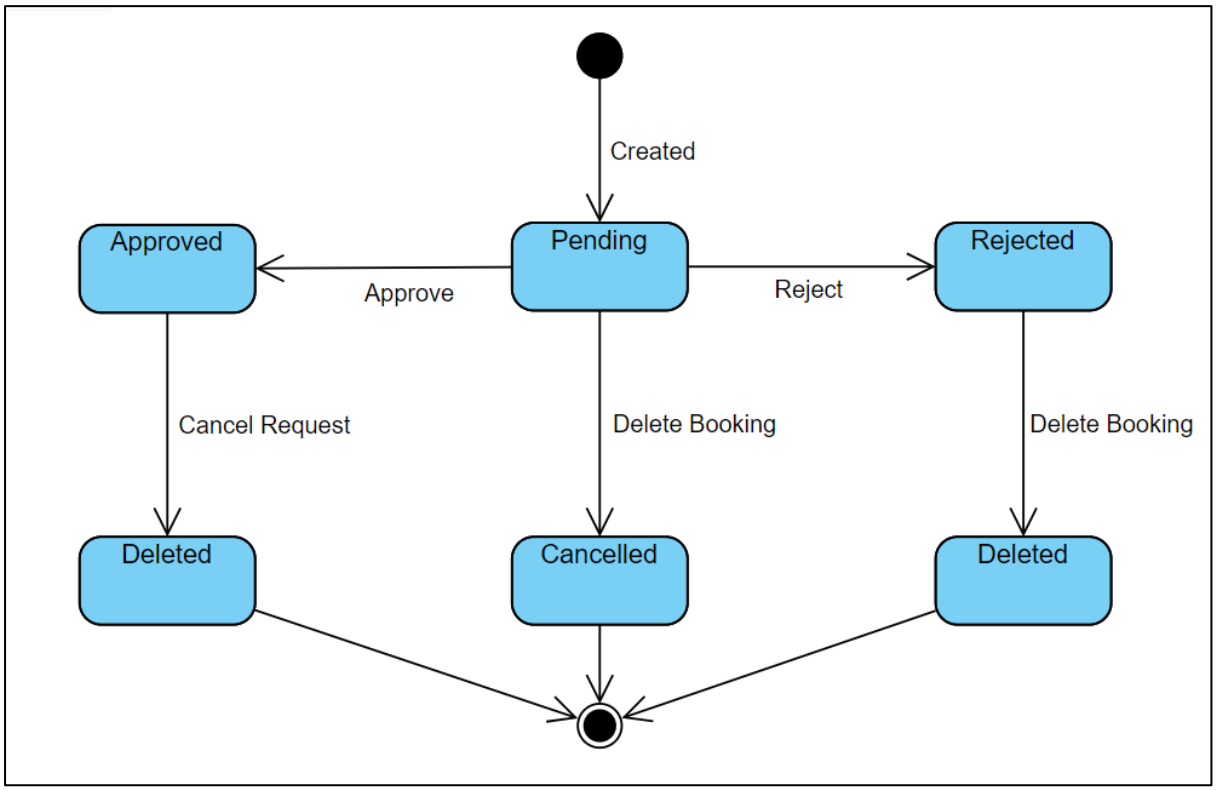

Figure 7.5 : State Diagram for Booking Entity

Ex. No: 8 **DATE : 17.10.2023**

SOFTWARE IMPLEMENTATION

Booking Form :

Backend API

```
import booking from "../models/BookingModel.js";
import halls from "../models/HallsModel.js";
import { autoInc } from "../utils/AutoIncrement.js";

//CREATE BOOKING
export const createBooking = async (req, res) => {
  const selectedHallName = req.body.Hall_Name;
  const data = await halls.findOne({ Hall_Name: selectedHallName });

  req.body.Faculty_ID = data.Faculty_ID;
  const bookingId = await autoInc();
  const newBooking = req.body;
  newBooking["Booking_ID"] = bookingId;
  console.log(newBooking);
  try {
    const savedBooking = await booking.create(newBooking);
    res.status(200).json(savedBooking);
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
    });
  }
};

//DELETE BOOKING
export const deleteBooking = async (req, res) => {
  try {
    const deleteBooking = await booking.findByIdAndDelete(req.params.id);
    res.status(200).json("Object has been deleted");
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
    });
  }
};
//GET BOOKING
export const getBooking = async (req, res) => {
  try {
    const hotel = await booking.findById(req.params.id);
    res.status(200).json(hotel);
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
```

```
    });
  }
};

//GET Users BOOKINGS
export const getUserBookings = async (req, res) => {
  try {
    const user = req.user;
    // const bookingdate = new Date(req.query.date)
    const userBookings = await booking.find({
      Student_ID: user.Student_ID,
    }); // , Date: {$gt : bookingdate}
    res.status(200).json(userBookings);
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
    });
  }
};

//GET Admin BOOKINGS
export const getAdminBookings = async (req, res) => {
  try {
    console.log(req.user);
    const halls = await booking.find({
      Faculty_ID: req.user.adminId,
    });
    res.status(200).json(halls);
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
    });
  }
};

//Update Bookings
export const updateBooking = async (req, res) => {
  try {
    const halls = await booking
      .find({
        Booking_ID: req.body.Booking_ID,
      })
      .updateOne({
        Status: req.body.Status,
      });
    res.status(200).json(halls);
  } catch (err) {
    res.status(400).json({
      status: "Failed",
      message: err,
    });
  }
};
```

```
104  //Get all Booknngs
105
106  export const getAllBookings = async (req, res) => {
107    try {
108      const halls = await booking.find();
109      res.status(200).json(halls);
110    } catch (err) {
111      res.status(400).json({
112        status: "Failed",
113        message: err,
114      });
115    }
116  };
117
118  export const getAvailableTimes = async (req, res, next) => {
119    try {
120      // Fetch approved bookings from MongoDB
121      const hallname = req.query.hallname;
122      const date = req.query.date;
123      const bookedSlots = await booking.find({
124        Status: "approved",
125        Hall_Name: hallname,
126        Date: date,
127      });
128
129      // Calculate available time slots
130      const openingTime = new Date(date); // Define your opening time //
131      openingTime.setHours(6, 0, 0, 0);
132
133      const closingTime = new Date(date); // Define your closing time //
134      closingTime.setHours(20, 30, 0, 0);
135      const timeSlots = [];
136
137      // Generate time slots between openingTime and closingTime
138      const currentTime = new Date(openingTime);
139
140      while (currentTime <= closingTime) {
141        timeSlots.push(new Date(currentTime));
142        currentTime.setMinutes(currentTime.getMinutes() + 30); // 30 minutes interval
143      }
144
145      // Remove booked slots from available time slots
146      const availableTimeSlots = timeSlots.filter((timeSlot) => {
147        const isOverlapping = bookedSlots.some((booking) => {
148          return timeSlot >= booking.Time_From && timeSlot < booking.Time_To;
149        });
150        return !isOverlapping;
151      });
152
153      res.json({ availableTimeSlots });
154    } catch (err) {
155      res.status(400).json({
156        status: "Failed",
157        message: err,
158      });
159    }
160  };
```

Model

```
import mongoose from "mongoose";

const bookingSchema = new mongoose.Schema(
  {
    Booking_ID: {
      type: "Number",
      required: true,
    },
    Faculty_ID: {
      type: "Number",
      required: true,
    },
    Hall_Name: {
      type: "String",
      required: true,
    },
    Student_ID: {
      type: "String",
      required: true,
    },
    Department: {
      type: "String",
      required: true,
    },
    Affiliated: {
      type: "String",
      required: true,
    },
    Status: {
      type: "String",
      enum: ["rejected", "approved", "pending"],
      default: "pending",
    },
    Date: {
      type: "Date",
      required: true,
    },
    Time_From: {
      type: "Date",
      required: false,
    },
    Time_To: {
      type: "Date",
      required: false,
    },
    Reason: {
      type: "String",
      required: true,
    },
    Remark: {
      type: "String",
      required: false,
    },
```

```
    },
    { timestamps: true }
);

export default mongoose.model("booking", bookingSchema);
```

Frontend Code

```
// import Datepicker from "tailwind-datepicker-react";
import { useEffect, useState } from "react";
import axios from "axios";

function StudentDashboardHallBookingBookingForm({ selectedHall }) {
  //GET HALLS FROM halls SCHEMA FROM MONGO
  const [halls, setHalls] = useState([]);
  const [affiliatedDept, setAffiliatedDept] = useState();
  const [Time_From, setTimeFrom] = useState("");
  const [Time_To, setTimeTo] = useState("");
  const [selectedDate, setSelectedDate] = useState("");
  const [reason, setReason] = useState();

  //
  const [showSuccessMessage, setShowSuccessMessage] = useState(false);
  const [showErrorMessage, setShowErrorMessage] = useState(false);

  //STUDENT ODA DEPARTMENT
  const [userData, setUserData] = useState("");
  useEffect(() => {
    const data = JSON.parse(localStorage.getItem("authToken"));
    setUserData(data);
  }, []);
  //

  useEffect(() => {
    axios
      .get("http://localhost:8800/api/halls")
      .then((response) => {
        setHalls(response.data);
      })
      .catch((error) => {
        console.error("Error fetching hall data:", error);
      });
  }, []);
  ///Handle Booking
  const handleBooking = async (event) => {
    event.preventDefault();

    try {
      const data = {
        Student_ID: userData.Student_ID,
        Hall_Name: selectedHall.Hall_Name,
        Department: userData.Department,
        Affiliated: affiliatedDept,
        Date: selectedDate,
```

```
47          Time_From: Time_From,
48          Time_To: Time_To,
49          Reason: reason,
50        };
51
52        const hallBooked = await fetch(
53          " http://localhost:8800/api/booking/createBooking",
54          {
55            method: "POST",
56            headers: {
57              "Content-Type": "application/json",
58              Authorization: `Bearer ${userData.token}`,
59            },
60            body: JSON.stringify(data),
61          }
62        );
63
64        if (hallBooked.status === 200) {
65          console.log("Booking created successfully");
66          setShowSuccessMessage(true); //SUCCESS MESSAGE
67          setShowErrorMessage(false);
68
69          event.target.reset();
70
71          setTimeout(() => {
72            setShowSuccessMessage(false);
73          }, 3000);
74        } else {
75          console.error("Failed to create booking");
76          setShowErrorMessage(true); //ERROR MESSAGE
77          setShowSuccessMessage(false);
78        }
79      } catch (error) {
80        console.error("Error creating booking:", error.message);
81        setShowErrorMessage(true);
82        setShowSuccessMessage(false);
```

```
    }
  };
  //

  //AVAILABLE SLOTS
  const [availableTimes, setAvailableTimes] = useState([]);

  useEffect(() => {
    if (selectedDate) {
      console.log("Fetching available time slots...");
      fetch(
        `http://localhost:8800/api/booking/availableslots?hallname=${selectedHall.Hall_Name}&date=${selectedDate}`
      )
        .then((response) => response.json())
        .then((data) => {
          const availableTimeSlots = data.availableTimeSlots.map(
            (timeStr) => new Date(timeStr)
          );
          setAvailableTimes(availableTimeSlots);
        });
    }
  }, [selectedDate]);

  useEffect(() => {
    // Set the initial value to the first element of availableTimes
    if (availableTimes.length > 0) {
      setTimeFrom(availableTimes[0]);
      setTimeTo(availableTimes[0]);
    }
  }, [availableTimes]);

  const handleTimeFromChange = (event) => {
    setTimeFrom(event.target.value);
  };

  const handleTimeToChange = (event) => {
    setTimeTo(event.target.value);
  };
  //
  const handleDateChange = (e) => {
    const selectedDate = e.target.value;
    setSelectedDate(selectedDate);
  };
  //

  return (
    <div className="sm:p-14 p-3 bg-zinc-100">
      <div className="text-sm sm:text-lg">
        Fill the following details and click submit to book the hall
      </div>
      {showSuccessMessage && (
        <div
          class="p-4 mb-4 text-sm text-green-800 rounded-lg bg-green-50 dark:bg-gray-800 dark:text-green-400"
          role="alert"
        >
          <span class="font-medium">Booking created successfully!</span>
        </div>
      )}

      {showErrorMessage && (
        <div
          class="p-4 mb-4 text-sm text-red-800 rounded-lg bg-red-50 dark:bg-gray-800 dark:text-red-400"
          role="alert"
        >
          <span class="font-medium">Failed to create Booking!</span> Try again
          later.
```

```
149             </div>
150           )}
151
152           <form className="py-10 sm:pr-20" onSubmit={handleBooking}>
153             <table className="table-auto w-full">
154               <tbody>
155                 <tr>
156                   <td className="w-1/6 sm:w-1/3 p-4">
157                     <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
158                       DEPARTMENT
159                       <label className="mx-3 font-bold">:</label>
160                     </label>
161                   </td>
162                   <td>
163                     <input
164                       type="text"
165                       value={userData.Department}
166                       readOnly
167                       className="bg-[#f8fafa] border border-gray-300 text-gray-900 text-md rounded-md
168                        focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
169                     />
170                   </td>
171                 </tr>
172                 <tr>
173                   <td className="w-1/6 sm:w-1/3 p-4">
174                     <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
175                       HALL FOR BOOKING
176                       <label className="mx-3 font-bold">:</label>
177                     </label>
178                   </td>
179                   <td>
180                     <input
181                       type="text"
182                       value={selectedHall.Hall_Name}
183                       readOnly
184                       className="bg-[#f8fafa] border border-gray-300 text-gray-900 text-md rounded-md
185                       focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
186                     />
187                   </td>
188                 </tr>
189                 <tr>
190                   <td className="w-1/6 sm:w-1/3 p-4">
191                     <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
192                       AFFILIATED DEPARTMENT/ CLUB
193                       <label className="mx-3 font-bold">:</label>
194                     </label>
195                   </td>
196                   <td>
197                     <input
198                       onChange={(e) => {
199                         setAffiliatedDept(e.target.value);
200                       }}
201                       className="bg-[#f8fafa] border border-gray-300 text-gray-900 text-md rounded-md
202                        focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
203                     />
204                   </td>
205                 </tr>
206                 <tr>
207                   <td className="w-1/6 sm:w-1/3 p-4">
208                     <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
209                       DATE
210                       <label className="mx-3 font-bold">:</label>
211                     </label>
212                   </td>
213                   <td>
214                     <input type="date" onChange={handleDateChange} />
215                   </td>
216                 </tr>
217                 <tr>
218                   <td className="w-1/6 sm:w-1/3 p-4">
```

```
                <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
                  TIME FROM
                  <label className="mx-3 font-bold">:</label>
                </label>
              </td>
              <td>
                <select
                  id="TimeFrom"
                  value={Time_From}
                  onChange={handleTimeFromChange}
                  className="bg-[#f8fafa] border border-gray-300 text-gray-900 text-md rounded-md
                   focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
                  required
                >
                  <option disabled value="">
                    Select a date
                  </option>
                  {availableTimes.map((time, index) => (
                    <option key={index} value={time}>
                      {time.toLocaleTimeString(undefined, {
                        hour: "2-digit",
                        minute: "2-digit",
                      })}
                    </option>
                  ))}
                </select>
              </td>
            </tr>
            <tr>
              <td className="w-1/6 sm:w-1/3 p-4">
                <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
                  TIME TO
                  <label className="mx-3 font-bold">:</label>
                </label>
              </td>
              <td>
                <select
                  id="TimeTo"
                  value={Time_To}
                  onChange={handleTimeToChange}
                  className="bg-[#f8fafa] border border-gray-300 text-gray-900 text-md rounded-md
                   focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
                  required
                >
                  <option disabled value="">
                    Select a date
                  </option>
                  {availableTimes.map((time, index) => (
                    <option key={index} value={time}>
                      {time.toLocaleTimeString(undefined, {
                        hour: "2-digit",
                        minute: "2-digit",
                      })}
                    </option>
                  ))}
                </select>
              </td>
            </tr>
            <tr>
              <td className="w-1/6 sm:w-1/3 p-4 align-top">
                <label className="text-sm sm:text-lg font-bold text-gray-900 flex justify-between">
                  REASON
                  <label className="mx-3 font-bold">:</label>
                </label>
              </td>
              <td className="pt-4">
                <input
                  onChange={(e) => {
                    setReason(e.target.value);
```

```
                    }}
                    className="bg-[#f8fafa] h-24 border border-gray-300 text-gray-900 text-md rounded-md
                     focus:ring-blue-500 focus:border-blue-500 block w-full p-1.5"
                  />
                </td>
              </tr>
            </tbody>
          </table>

          <button
            type="submit"
            className="text-white bg-sky-500 hover:bg-sky-600 focus:ring-4 focus:outline-none
             focus:ring-blue-300 font-medium mt-5 rounded-md text-sm w-full sm:w-auto px-5 py-2.5 text-center"
          >
            Book Hall
          </button>
        </form>
      </div>
    );
}

export default StudentDashboardHallBookingBookingForm;

```

Ex. No: 9 **DATE : 14.11.2023**

CONCLUSION

In conclusion, the implementation of a campus hall booking system for software development record brings about several significant advantages for both students and administrators. This system streamlines the process of reserving campus halls for software development purposes, providing a more efficient and organized approach to managing these valuable resources.

Moreover, the system facilitates easy access to available halls, reducing the time and effort required for students to find suitable spaces for their software development projects. This, in turn, enhances productivity and allows students to focus more on their coding and collaborative work rather than logistical challenges.

In conclusion, the campus hall booking system for stands as a practical solution that aligns with the evolving needs of educational institutions. Its implementation not only enhances the student experience by simplifying the booking process but also empowers administrators with valuable data for informed decision-making. Overall, this system contributes to a more efficient, transparent, and well-managed environment for software development activities within the campus community.